The Bell System Technical Journal
Vol. 51, No. 2, February, 1972
Printed in U.S.A.

On Finding the Paths Through a Network

By N. J. A. SLOANE

(Manuscript received May 19, 1971)

Given a directed graph G, algorithms are discussed for finding (i) all paths through G with prescribed originating and terminating nodes, (ii) a subset of these paths containing all the edges, (iii) a subset containing all the edge-edge transitions, and (iv) a subset containing the most likely paths.

I. INTRODUCTION

Informally, a *directed graph* consists of a set of vertices or *nodes* together with a set of directed *edges* joining the nodes. (All of the figures below show directed graphs; for a formal definition see page 10 of Ref. 1. There may be more than one edge with the same originating and terminating nodes, and the originating and terminating nodes of an edge may coincide.)

Common examples of directed graphs are state diagrams of systems: the nodes represent states of the system and an edge directed from node N_i to node N_j means that it is possible for the system to go directly from state N_i to state N_j.

The following questions concerning the paths through a directed graph arose in testing for possible errors sections of the stored program of a No. 1 ESS electronic switching system.[2] However, these questions and the algorithms for their solution seem of sufficient general interest to warrant stating them independently of their origin.

Given a directed graph G, the questions are: (*i*) Find the set α of all paths through G with prescribed originating and terminating nodes. (A path is just what one would expect; a formal definition is given in Section II.) (*ii*) Find a small subset of α which contains every edge occurring in α. (*iii*) Find a small subset of α which contains all the edge-edge transitions occurring in any path in α. (*iv*) If a probability measure is associated with the edges of G, find the most probable paths in α.

These questions and algorithms for their solution are discussed in Sections III, V, VI, and VII, respectively. Section II is concerned with

the notation used to describe paths, and Section IV with an algorithm for partially solving a combinatorial problem encountered in Sections V and VI.

II. NOTATION FOR PATHS

Definition: A *path* from node N_1 to N_2 in a directed graph is a sequence of (not necessarily distinct) edges $e_1, e_2, \cdots, e_\ell$ with the property that there are nodes $N_1 = n_1, n_2, \cdots, n_{\ell+1} = N_2$ such that e_i is directed from n_i to n_{i+1} for $i = 1, 2, \cdots, \ell$. The *length* ℓ of a path is the number of edges it contains.

A path is specified by giving the ordered *string* $e_1e_2 \cdots e_\ell$ of its edges. (We are in fact describing paths by the notation used in automata theory to describe regular expressions, as given, for example, in Ref. 3 and chapter 5 of Ref. 4. However, the treatment given here is self-contained.)

It is convenient to include in the definition a path of zero length (whose endpoints N_1 and N_2 must coincide). This path is specified by the *empty string* Λ (not to be confused with the empty set ϕ).

A collection of paths is specified by the sum of the strings of the individual paths.

If S is a string, S^i denotes $SS \cdots S$ (i.e., S concatenated i times) and S^* denotes $\Lambda + S + S^2 + S^3 + \cdots$. For example, in Fig. 1 the collection of all paths from

N_2 to N_1 is ϕ,

N_1 to N_1 is Λ,

N_1 to N_2 is a,

N_4 to N_4 is $\Lambda + f + f^2 + f^3 + \cdots = f^*$,

N_2 to N_3 is $d + ce + cfe + cf^2e + \cdots = d + cf^*e$,

N_1 to N_3 is $ad + (ac + b)f^*e$.

Parentheses are used in the natural way. The following rules are easily verified. Here S is any sum of strings.

$\phi + S = S, \phi S = S\phi = \phi$

$S^* = \Lambda + S + S^2 + S^3 + \cdots$

$\Lambda^* = \Lambda$

$\Lambda S = S$

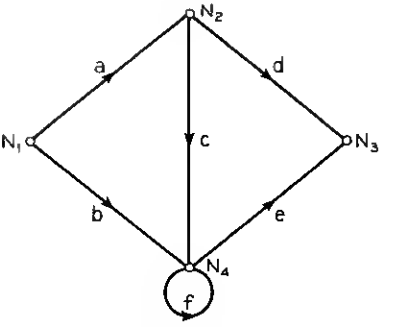


Fig. 1—An example.

$$(\Lambda + S)^* = \Lambda + SS^* = \Lambda + S^* = S^*$$

$$S_1 + S_2S_2^*S_1 = S_2^*S_1\,;\; S_1 + S_1S_2^*S_2 = S_1S_2^*$$

$$(S_1 + S_2)^* = (S_1^* + S_2^*)^*.$$

III. FINDING ALL PATHS THROUGH A GRAPH

Let G be a directed graph with n nodes labeled N_1 , N_2 , $\cdots$, N_n . Methods are given for finding all paths through G having prescribed starting node N_μ and (not necessarily distinct) terminating node N_ν . We first describe the McNaughton–Yamada Algorithm, which requires on the order of n^3 steps.

Definition: Let α_{ij}^k denote the set of paths which start at N_i , end at N_j , and do not pass through any intermediate node N_p with $p > k$, for $k = 0, 1, \cdots, n$, and $i, j = 1, 2, \cdots, n$.

The algorithm successively computes α_{ij}^0 for all i and j, then α_{ij}^1 for all i and j, $\cdots$, then α_{ij}^{n-1} for all i and j. The final step is to compute $\alpha_{\mu\nu}^n$, the set of all paths from N_μ to N_ν with no restriction on intermediate nodes, which is the desired result.

The inductive step proceeds as follows. Suppose $\alpha_{i,j}^{k-1}$ is known for all i, j, and we wish to obtain $\alpha_{i,j}^k$. Referring to Fig. 2, we see that the fundamental recurrence equation is

$$\alpha_{ij}^k = \alpha_{ij}^{k-1} + \alpha_{ik}^{k-1}(\alpha_{kk}^{k-1})^*\alpha_{kj}^{k-1}. \tag{1}$$

In words, this says that the paths from N_i to N_j containing intermediate nodes as high as k are made up of those containing intermediate nodes only as high as $k - 1$, α_{ij}^{k-1}, plus all possible paths containing N_k as an intermediate node, $\alpha_{ik}^{k-1}(\alpha_{kk}^{k-1})^*\alpha_{kj}^{k-1}$. When k is equal to either i or j,

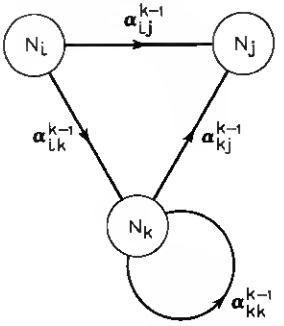


Fig. 2—The inductive step.

(1) may be simplified. We now give the complete statement of the algorithm.

THE MCNAUGHTON–YAMADA ALGORITHM[3]

1. *The Initial Step*

Define α_{ij}^0 for all $i, j = 1, \cdots, n$ by:

(1.1) if $i \neq j$,

$$\alpha_{ij}^0 = \begin{cases} \phi \text{ if there is no edge from } N_i \text{ to } N_j, \\ e_1 + e_2 + \cdots \text{ if edges labeled } e_1, e_2, \cdots \text{ join } N_i \text{ to } N_j; \end{cases}$$

(1.2) if $i = j$,

$$\alpha_{ii}^0 = \begin{cases} \Lambda \text{ if there is no edge from } N_i \text{ to itself,} \\ \Lambda + e_1 + e_2 + \cdots \text{ if edges labeled } e_1, e_2, \cdots \text{ join } N_i \text{ to itself.} \end{cases}$$

2. *The Inductive Step* (Refer to Fig. 2)

For $k = 1, 2, \cdots, n-1$ compute α_{ij}^k for all $i, j = 1, 2, \cdots, n$ from:

(2.1) if $k \neq i$, $k \neq j$ then

$$\alpha_{ij}^k = \alpha_{ij}^{k-1} + \alpha_{ik}^{k-1}(\alpha_{kk}^{k-1})^* \alpha_{kj}^{k-1};$$

(2.2) if $i \neq j$ and $k = i$,

$$\alpha_{ij}^i = (\alpha_{ii}^{i-1})^* \alpha_{ij}^{i-1};$$

(2.3) if $i \neq j$ and $k = j$,

$$\alpha_{ij}^j = \alpha_{ij}^{j-1}(\alpha_{jj}^{j-1})^*;$$

(2.4) if $i = j = k$,

$$\alpha_{ii}^i = (\alpha_{ii}^{i-1})^*.$$

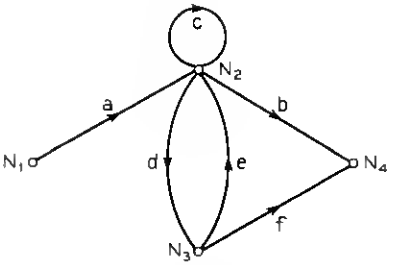


Fig. 3—An example.

3. *The Final Step*

Finally, use whichever of (2.1) to (2.4) is appropriate to calculate $\alpha^n_{\mu,\nu}$, the set of all paths from N_μ to N_ν.

Remark: In steps 2 and 3, after obtaining expressions of the form $\alpha_{\text{new}} = \cdots (\beta)^* \cdots$, it may be convenient to simplify $(\beta)^*$ by means of the rules given at the end of Section II.

An Example: We will use the McNaughton–Yamada algorithm to compute the set of all paths in Fig. 3 which start at N_1 and end at N_4, or, in other words, α^4_{14}.

Step 1.

		j			
	i	1	2	3	4
α^0_{ij}	1	Λ	a	ϕ	ϕ
	2	ϕ	$\Lambda + c$	d	b
	3	ϕ	e	Λ	f
	4	ϕ	ϕ	ϕ	Λ

Step 2. Since there are no paths into N_1, $\alpha^1_{ij} = \alpha^0_{ij}$ for all i, j.

		j			
	i	1	2	3	4
α^2_{ij}	1	Λ	$a(\Lambda + c)^*$	$a(\Lambda + c)^*d$	$a(\Lambda + c)^*d$
	2	ϕ	$(\Lambda + c)^*$	$(\Lambda + c)^*d$	$(\Lambda + c)^*b$
	3	ϕ	$e(\Lambda + c)^*$	$\Lambda + e(\Lambda + c)^*d$	$f + e(\Lambda + c)^*b$
	4	ϕ	ϕ	ϕ	Λ

where we have used the rules $\phi + S = S$ and $\phi S = S\phi = \phi$. This may be further simplified using the rules at the end of Section II as follows.

α_{ij}^2

i \ j	1	2	3	4
1	Λ	ac^*	ac^*d	ac^*b
2	ϕ	c^*	c^*d	c^*b
3	ϕ	ec^*	$\Lambda + ec^*d$	$f + ec^*b$
4	ϕ	ϕ	ϕ	Λ

Since there are no paths out of N_4, $\alpha_{ij}^4 = \alpha_{ij}^3$ for all i, j. We can therefore go directly to Step 3:

$$\begin{aligned}\alpha_{14}^4 = \alpha_{14}^3 &= \alpha_{14}^2 + \alpha_{13}^2(\alpha_{33}^2)^*\alpha_{34}^2 \\ &= ac^*b + ac^*d(\Lambda + ec^*d)^*(f + ec^*b) \\ &= ac^*b + ac^*d(ec^*d)^*(f + ec^*b),\end{aligned}$$

which, if required, can be expanded to give

$$\begin{aligned}\alpha_{14}^4 = {} & ab + acb + ac^2b + \cdots \\ & + adf + adeb + adecb + adec^2b + \cdots \\ & + adedf + adedeb + adedecb + \cdots \\ & + adecdf + adecdeb + adecdecb + \cdots \\ & + acdf + acdeb + acdecb + acdec^2b + \cdots \\ & + acdedf + acdedeb + acdedecb + \cdots \\ & + \cdots .\end{aligned}$$

It may be verified that this includes all possible paths from N_1 to N_4.

Remarks: (*i*) When programmed in a computer language capable of handling strings, such as SNOBOL4,[5] this algorithm involves the calculation of n $n \times n$ matrices (requiring on the order of n^3 steps). Enough storage space is required to hold two $n \times n$ matrices (the current $[\alpha_{ij}^k]$, $i, j = 1, \cdots, n$, matrix and the previously calculated $[\alpha_{ij}^{k-1}]$, $i, j = 1, \cdots, n$, matrix) each entry of which is a string of letters, parentheses, +'s and *'s. (*ii*) With very little extra work Step 3 can be modified to give the paths between several pairs of nodes. This is valuable for analyzing large graphs, as we now show.

Analysis of Large Graphs by Partitioning

Since the time required for the McNaughton–Yamada algorithm grows as the cube of the number of states, large graphs cannot be handled directly. However, such graphs can usually be handled by partitioning them into smaller subgraphs, applying the algorithm to each subgraph separately, and then reapplying the algorithm to the network of subgraphs. The following simple example will illustrate the method.

Figure 4 shows a graph G partitioned into two subgraphs G_1 and G_2 which are interconnected at nodes N_2 and N_3. (Only edges between the subgraphs are shown.) Suppose we wish to find all paths from N_1 to N_4. If G_1, G_2 each contain 20 nodes, a direct application of the McNaughton–Yamada algorithm would require on the order of $40^3 = 64{,}000$ steps. This number is considerably reduced by the following technique.

Let $\beta_{ij}(G_\nu)$ denote the set of all paths starting at N_i, ending at N_j, and lying entirely in the subgraph G_ν.

We first apply the McNaughton–Yamada algorithm to G_1 and G_2 to obtain $\beta_{ij}(G_1)$, $i, j = 1, 2$, and $\beta_{ij}(G_2)$, $i, j = 3, 4$. That is, we first find all the paths between the interconnecting nodes that lie completely in one of the subgraphs. (This will take on the order of $2 \cdot 20^3 = 16{,}000$ steps.)

We now replace G by the condensed graph $\bar{G}$ of Fig. 5. $\bar{G}$ contains (*i*) nodes $\bar{N}_1$, $\bar{N}_4$ corresponding to the terminal nodes N_1, N_4, (*ii*) nodes $\bar{N}_2$, $\bar{N}_3$ corresponding to the interconnecting nodes N_2, N_3, (*iii*) edges a, b corresponding to the interconnecting edges a, b of G, and (*iv*) edges corresponding to all the paths $\beta_{ij}(G_1)$, $i, j = 1, 2$, and $\beta_{ij}(G_2)$, $i, j = 3, 4$, in G.

The McNaughton–Yamada algorithm is now used to obtain all paths from $\bar{N}_1$ to $\bar{N}_4$ in $\bar{G}$. (This takes on the order of $4^3 = 64$ steps.) It is clear that these paths are exactly all the paths from N_1 to N_4 in the original graph G. Partitioning into two equal subgraphs has thus reduced the number of steps by approximately a factor of four. (Parti-

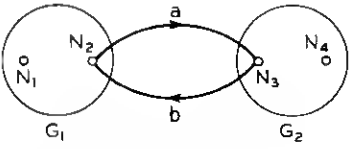


Fig. 4—A graph partitioned into subgraphs.

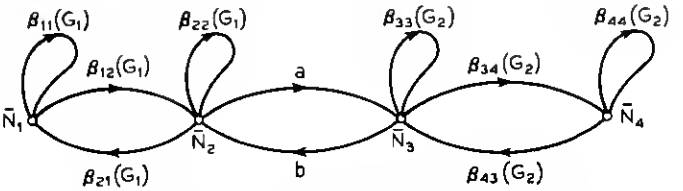


Fig. 5—The condensed graph corresponding to Fig. 4.

tioning into k equal subgraphs would reduce it by a factor of about k^2.)

The general method of analyzing a large graph by partitioning should now be clear.

If n_o is the largest number of nodes that can be directly handled by the McNaughton–Yamada algorithm, then it is desirable to partition G in such a way that no subgraph has more than n_o nodes, and that the total number of interconnecting nodes (which is the number of nodes in the condensed graph) is also less than n_o. (Of course the sugbraphs may themselves be partitioned.)

IV. THE COVERING PROBLEM

In Sections V and VI we will encounter a basic combinatorial problem, the covering problem, which may be stated as follows. Suppose a set $S = \{s_1, s_2, \cdots, s_n\}$ of n elements is given, together with a family $\mathfrak{F}$ of subsets of S,

$$\mathfrak{F} = \{X_1, X_2, \cdots, X_m\}, \qquad X_i \subseteq S.$$

The problem is to find a subfamily $\mathfrak{IC} \subseteq \mathfrak{F}$, say

$$\mathfrak{IC} = \{X_{i_1}, X_{i_2}, \cdots, X_{i_l}\},$$

where ℓ is as small as possible, such that every element of S appearing in $\mathfrak{F}$ also appears in $\mathfrak{IC}$, or formally, such that

$$X_1 \cup X_2 \cup \cdots \cup X_m = X_{i_1} \cup X_{i_2} \cup \cdots \cup X_{i_l}.$$

$\mathfrak{IC}$ is called a *covering set* for $\mathfrak{F}$.

The family $\mathfrak{F}$ may be represented by an $m \times n$ (0, 1) matrix $\mathfrak{M} = (m_{ij})$, where

$$m_{ij} = 1 \quad \text{if} \quad s_j \in X_i,$$
$$0 \quad \text{if} \quad s_j \notin X_i.$$

The ith row of $\mathfrak{M}$, written $I(X_i)$, is called the *indicator vector* of X_i, since it indicates which elements of S belong to X_i.

The problem is to find a minimal set of rows which together contain a 1 in every nonzero column. Equivalently, if we relabel the matrix so that columns correspond to subsets and rows to elements, the problem is to find a minimal system of representatives for the subsets. This problem is known to be difficult (Ref. 6, page 521).

The direct attack is to look at the rows taken 1, 2, 3, $\cdots$, m at a time, until a covering set is found; this finds a minimal covering set, but may take up to $2^m - 1$ steps. Several methods[7-13] have been given which are faster than the direct attack, but are still impractical for large m. Roth's algorithm[14] finds a locally minimal cover which has a high probability of being the minimal cover, for quite large values of m (up to several hundred).

However, for our purposes, the following extremely simple (and appropriately named) algorithm is adequate. It finds a covering set in at most $\frac{1}{2}m^2$ steps, but may not find a minimal cover.

THE GREEDY ALGORITHM

The algorithm proceeds inductively, starting with $\mathfrak{C} = \phi$ and (greedily) *adding to* $\mathfrak{C}$, *each time that particular* X_i *which will contribute the greatest number of new elements.*

We keep track of the elements in $\mathfrak{C}$ at each step by means of the indicator vector

$$I(\mathfrak{C}) = I(\bigcup_{X \in \mathfrak{C}} X)$$

and stop when this is equal to

$$I(\mathfrak{F}) = I(\bigcup_{X \in \mathfrak{F}} X).$$

1. *The Initial Step*

Set $\mathfrak{C} = \phi$, $I(\mathfrak{C}) = (0, 0, \cdots, 0)$.

2. *The Inductive Step*

Search through all $X_i \in \mathfrak{F}$ that are not in $\mathfrak{C}$ and find an X_k which maximizes the number of elements of S which are in X_k but not in $\mathfrak{C}$, i.e., which maximizes weight $(I(X_k)$. AND. NOT. $I(\mathfrak{C}))$. (The weight of a vector is the number of its nonzero components, $(a_1, \cdots, a_n)$. AND. $(b_1, \cdots, b_n) = (a_1$ AND $b_1, \cdots, a_n$ AND $b_n)$, NOT. $(a_1, \cdots, a_n) =$ (NOT $a_1, \cdots,$ NOT $a_n)$, and .OR. is defined similarly.) Break ties in any way.

Add X_k to $\mathfrak{C}$, and calculate the new $I(\mathfrak{C})$ = old $I(\mathfrak{C})$. OR. $I(X_k)$. Repeat Step 2 until $I(\mathfrak{C}) = I(\mathfrak{F})$; then stop.

Remarks:

(*i*) The greedy algorithm often finds a covering set which is close to minimal, although it is possible to construct examples when the minimal covering set contains two subsets while the greedy algorithm uses more than N subsets, for any preassigned value of N. Are such examples rare? The behavior of the algorithm for a random family $\mathfrak{F}$ seems to be unknown.

(*ii*) Since the algorithm involves simple calculations with binary vectors it may be easily programmed on a computer.

Example 1: The set of all paths from N_1 to N_4 in Fig. 6 consists of

$$(a_1 + a_2)(b_1 + b_2)(c_1 + c_2) = a_1b_1c_1 + a_1b_1c_2 + a_1b_2c_1 + a_1b_2c_2 + a_2b_1c_1 + a_2b_1c_2 + a_2b_2c_1 + a_2b_2c_2 .$$

Suppose it is desired to find a minimal subset of these paths which contains all the edges $S = \{a_1 , a_2 , b_1 , b_2 , c_1 , c_2\}$. $\mathfrak{F}$ consists of the following eight subsets of S, shown together with their indicator vectors.

i	X_i	$I(X_i)$
1	$a_1b_1c_1$	1 0 1 0 1 0
2	$a_1b_1c_2$	1 0 1 0 0 1
3	$a_1b_2c_1$	1 0 0 1 1 0
4	$a_1b_2c_2$	1 0 0 1 0 1
5	$a_2b_1c_1$	0 1 1 0 1 0
6	$a_2b_1c_2$	0 1 1 0 0 1
7	$a_2b_2c_1$	0 1 0 1 1 0
8	$a_2b_2c_2$	0 1 0 1 0 1

The greedy algorithm then proceeds as follows.

Step 1. $\mathfrak{C} = \phi$, $I(\mathfrak{C}) = 000000$.

Step 2. Weight $(I(X_i).$ AND. $111111) = 3$ for all i, so we pick X_1 (any X_i will do) and add it to $\mathfrak{C}$: $\mathfrak{C} = \{X_1\}$, $I(\mathfrak{C}) = 101010$.

Step 2 again. Weight $(I(X_i).$ AND. $010101)$ is maximized by $i = 8$.

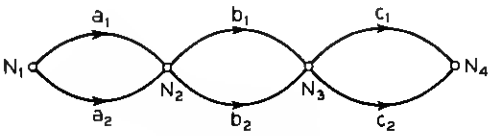


Fig. 6—An example.

Then $\mathfrak{C} = \{X_1 , X_8\}$, $I(\mathfrak{C}) = 101010$. OR. $010101 = 111111$. The algorithm terminates having found

$$\mathfrak{C} = \{a_1b_1c_1 , a_2b_2c_2\}$$

which is a correct solution.

Example 2: The greedy algorithm does not always find a minimal covering set, as the following example shows.

$$S = \{1, 2, 3, 4, 5, 6\}$$

$$\mathfrak{F} = \{X_1 = \{1, 2, 3\}, X_2 = \{4, 5, 6\}, X_3 = \{1, 3, 4, 6\}.$$

The greedy algorithm finds $\mathfrak{C} = \{X_1 , X_2 , X_3\}$, while the minimal $\mathfrak{C}$ is $\{X_1 , X_2\}$.

V. FINDING A SMALL SET OF PATHS CONTAINING ALL EDGES

As before, let G be a directed graph with nodes $N_1 , N_2 , \cdots , N_n$. Let $\alpha_{\mu\nu}$ denote the set of paths from N_μ to N_ν.

Definition: A set $\beta_{\mu\nu}$ of paths from N_μ to N_ν is said to be a *spanning set* if every edge occurring in the set $\alpha_{\mu\nu}$ occurs in $\beta_{\mu\nu}$.

Example: In Fig. 7, the set of all paths from N_1 to N_2 is

$$\alpha_{12} = (a + b)(c + d) = ac + ad + bc + bd,$$

whereas an example of a spanning set is $\beta_{12} = ac + bd$.

The problem we consider in this section is to find a small spanning set $\beta_{\mu\nu}$. Finding a *minimal* spanning set appears difficult, and the only method we know is essentially an exhaustive search, as given in the next paragraph. The main algorithm of this section, algorithm B, gives a small spanning set $\beta_{\mu\nu}$ with a reasonable amount of computation.

Finding the Smallest Spanning Set $\beta_{\mu\nu}$ by Exhaustive Search

This may be accomplished by first applying the McNaughton–Yamada algorithm of Section III to produce a condensed list of all paths from N_μ to N_ν. Then truncate each expression S^* appearing in this list to $\Lambda + S$. (Since there is no need to go around a loop more than once in

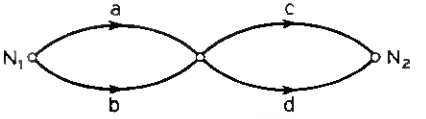


Fig. 7—An example.

succession, we can throw away the remaining terms of $S^* = \Lambda + S + S^2 + S^3 + \cdots$.) We now have a *finite* spanning set $\beta_{\mu\nu}$ and can use an exhaustive search to get a minimal set.

The difficulty with this method is that the number of terms obtained in the final list will be very large. To illustrate we apply the method to the four-node graph of Fig. 3. We found that the complete set of paths from N_1 to N_4 is

$$\alpha_{14} = ac^*b + ac^*d(ec^*d)^*(f + ec^*b).$$

Truncating each S^* to $\Lambda + S$, we obtain

$$a(\Lambda + c)b + a(\Lambda + c)d(\Lambda + e(\Lambda + c)d)(f + e(\Lambda + c)b),$$

which, when parentheses are removed, becomes

$$ab + acb + adf + adeb + adecb + adedf + adedeb + adedecb$$
$$+ adecdf + adecdeb + adecdecb + acdf + aedeb + acdecb + acdedf$$
$$+ acdedeb + acdedecb + acdecdf + acdecdeb + acdecdecb.$$

Then by inspection, or from the greedy algorithm of Section IV, we find that a minimal spanning set is for example

$$\beta_{14} = adf + adecb.$$

An Approximate Solution to the Problem–Algorithm B

We noticed in the above example that the difficulty was not in finding a minimal spanning set—indeed there are a large number of ways of choosing one—but rather in the very rapid increase in the number of terms to be handled. The algorithm to be described now keeps the lists involved small.

The basic idea is to follow the McNaughton–Yamada algorithm, but to use the greedy algorithm *twice at each step to reduce the complete path sets α_{ij}^k to small covering sets β_{ij}^k*.

Definition: Let β_{ij}^k be a set of paths from N_i to N_j containing no internal node N_p with $p > k$ and containing every edge appearing in α_{ij}^k.

Then $\beta_{\mu\nu}^n = \beta_{\mu\nu}$ is an example of a spanning set of paths from N_μ to N_ν, which is what we are seeking.

The algorithm will form the β_{ij}^k by induction on k. At each step we will keep a record of the edges in β_{ij}^k by means of its indicator vector $I(\beta_{ij}^k)$.

The inductive step proceeds as follows. Suppose β_{ij}^{k-1} is known for all i, j, and we wish to obtain β_{ij}^k (see Fig. 8). We restrict ourselves here

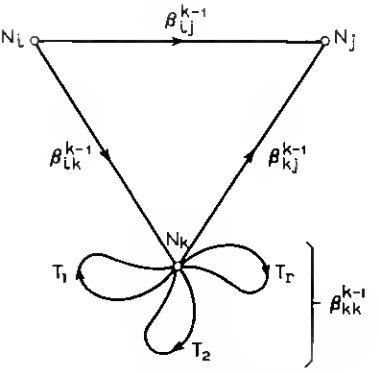


Fig. 8—The inductive step of algorithm B.

to the case when i, j and k are distinct, the other cases being left to the detailed statement of the algorithm.

Suppose $\beta_{kk}^{k-1} = T_1 + T_2 + \cdots + T_r$, where each T_a is a path from N_k to N_k. Then a possible choice for β_{ij}^{k} is

$$\beta_{ij}^{k-1} + \beta_{ik}^{k-1} T_{i_1} T_{i_2} \cdots T_{i_l} \beta_{kj}^{k-1} \tag{2}$$

where we have used just enough T_{i_r}'s to include all the edges in $T_1 + T_2 + \cdots + T_r$ that were not already contained in

$$\beta_{ij}^{k-1} + \beta_{ik}^{k-1} \beta_{kj}^{k-1}.$$

A better choice for β_{ij}^{k}, however, is to obtain (2) and then find a small spanning subset of (2) by the greedy algorithm.

We now give the algorithm.

ALGORITHM B

Each β_{ij}^{k} will have the form of a sum of strings of edges, without *'s or parentheses.

1. *The Initial Step*

Define β_{ij}^{0} for all $i, j = 1, \cdots, n$ by:

(1.1) if $i \neq j$,

$$\beta_{ij}^{0} = \begin{cases} \phi \text{ if there is no edge from } N_i \text{ to } N_j, \\ c_1 + c_2 + \cdots \text{ if edges labeled } c_1, c_2, \cdots \text{ join } N_i \text{ to } N_j; \end{cases}$$

(1.2) if $i = j$,

$$\beta_{ii}^{0} = \begin{cases} \Lambda \text{ if there is no edge from } N_i \text{ to itself,} \\ c_1 c_2 \cdots \text{ if edges labeled } c_1, c_2, \cdots \text{ join } N_i \text{ to itself.} \end{cases}$$

2. *The Inductive Step* (Refer to Fig. 8)

For $k = 1, 2, \cdots, n-1$, compute β_{ij}^k for all $i, j = 1, 2, \cdots, n$ as follows:

(2.1) If $k \neq i$, $k \neq j$:

(2.1.1) Let the terms of β_{kk}^{k-1} be

$$\beta_{kk}^{k-1} = T_1 + T_2 + \cdots + T_r .$$

(2.1.2) Form the indicator vector

$$I_1 = I(\beta_{ij}^{k-1}) .\text{OR}. I(\beta_{ik}^{k-1}) .\text{OR}. I(\beta_{kj}^{k-1}). \tag{2.1.3}$$

(This includes all the edges in the three sides of the triangle of Fig. 8.)
(2.1.4) Using the greedy algorithm, find a small subset of the T_a's in (2.1.1) which contains all the edges in

$$I(\beta_{kk}^{k-1}) .\text{AND. NOT. } I_1 ,$$

i.e., find a small subset of the terms T_a which includes all the new edges they contain. Let this subset be $T_{a_1} + T_{a_2} + \cdots + T_{a_m}$.

(2.1.5) Form the set

$$\beta_{ij}^{k-1} + \beta_{ik}^{k-1} T_{a_1} T_{a_2} \cdots T_{a_m} \beta_{kj}^{k-1} \cdots \tag{2.1.6}$$

(By construction, this now contains all the edges visible in Fig. 8.)
(2.1.7) Apply the greedy algorithm to the set (2.1.6) to find a small spanning subset. This is β_{ij}^k.

(2.2) If $i \neq j$ and $k = i$, replace (2.1.3) by $I_1 + I(\beta_{ij}^{i-1})$, and replace (2.1.6) by

$$T_{a_1} T_{a_2} \cdots T_{a_m} \beta_{ij}^{i-1}.$$

(2.3) If $i \neq j$ and $k = j$, replace (2.1.3) by $I_1 = I(\beta_{ij}^{j-1})$, and replace (2.1.6) by

$$\beta_{ij}^{j-1} T_{a_1} T_{a_2} \cdots T_{a_m} .$$

(2.4) If $i = j = k$:
Replace (2.1.3) by $I_1 = 0$ and replace steps (2.1.5) and (2.1.7) by

$$\beta_{kk}^k = T_{a_1} T_{a_2} \cdots T_{a_m} .$$

3. *The Final Step*

Use whichever of (2.1) to (2.4) is appropriate to calculate $\beta_{\mu\nu}^n$, the desired result.

Example: We use algorithm B to obtain a small spanning set β_{14} of paths from node 1 to node 4 in Fig. 3.

$\beta_{ij}^0 = \beta_{ij}^1$

	j			
i	1	2	3	4
1	Λ	a	ϕ	ϕ
2	ϕ	c	d	b
3	ϕ	e	Λ	f
4	ϕ	ϕ	ϕ	Λ

β_{ij}^2

	j			
i	1	2	3	4
1	Λ	ac	acd	acb
2	ϕ	c	cd	cb
3	ϕ	ec	ecd	$f + ecb$
4	ϕ	ϕ	ϕ	Λ

The last step will be shown in detail.

(2.1.1) $\beta_{33}^2 = ecd = T_1$.

(2.1.2) $I_1 = I(\beta_{14}^2)$. OR. $I(\beta_{13}^2)$. OR. $I(\beta_{34}^2) = 111000$. OR. 101100. OR. $011011 = 111111$.

(2.1.4) NOT. $I_1 = 000000$ so no T_a's need be used.

(2.1.5) $\beta_{14}^2 + \beta_{13}^2\beta_{34}^2 = acb + acd(f + ecb) = acb + acdf + acdecb$.

(2.1.7) From the greedy algorithm, $\beta_{14}^3 = \beta_{14}^4 = acdecb + acdf$, which is a minimal solution (although minimal solutions with shorter strings are possible, such as $acdeb + adf$).

Remarks: (*i*) If a fast version of the greedy algorithm is available, the computation time for algorithm B should not be much more than for the McNaughton–Yamada algorithm. (*ii*) An edge forming a loop of length one may be deleted from any sum of strings in which it appears more than once. If there are many such edges the algorithm should be modified to make a list of such edges and periodically delete duplicates from the β_{ij}^k . The modified algorithm would then give the improved solution $acdeb + adf$ to the above example. (*iii*) As in Section III, large networks may be handled by partitioning.

VI. FINDING A SMALL SET OF PATHS CONTAINING ALL EDGE-EDGE TRANSITIONS.

With $\alpha_{\mu\nu}$ defined as before, in this section we consider the problem of finding a small subset $\gamma_{\mu\nu}$ of $\alpha_{\mu\nu}$ with the property that every edge-edge transition appearing in any path from N_μ to N_ν appears in $\gamma_{\mu\nu}$.

For example, consider the graph of Fig. 6. Here the set of all paths from N_1 to N_4 is

$$\begin{aligned}\alpha_{14} &= (a_1 + a_2)(b_1 + b_2)(c_1 + c_2) \\ &= a_1b_1c_1 + a_1b_1c_2 + a_1b_2c_1 + a_1b_2c_2 \\ &\quad + a_2b_1c_1 + a_2b_1c_2 + a_2b_2c_1 + a_2b_2c_2 ,\end{aligned}$$

and an example of γ_{14} is

$$\gamma_{14} = a_1b_1c_1 + a_1b_2c_1 + a_2b_1c_2 + a_2b_2c_2 .$$

To check this we observe that α_{14} contains eight distinct edge-edge transitions:

$$a_1b_1 , a_1b_2 , a_2b_1 , a_2b_2 , b_1c_1 , b_1c_2 , b_2c_1 , b_2c_2 ,$$

and all of these appear in γ_{14} .

Of course γ_{14} is not unique, another example being

$$a_1b_1c_2 + a_1b_2c_2 + a_2b_1c_1 + a_2b_2c_1 .$$

The idea of the solution is to construct from G a new graph called the transition graph, G^T, which will have an edge for every edge-edge transition in G, and then to apply algorithm B to G^T.

Suppose then that G is given and it is desired to find $\gamma_{\mu\nu}$. First form the augmented graph $\bar{G}$ by adding to G a node N_0 which is connected to N_μ by an edge z_1 , and a node N_{n+1} to which N_ν is connected by an edge z_2 (see Fig. 9).

From $\bar{G}$ we construct the transition graph G^T as follows. The nodes of G^T are (i) a node denoted (N_0), and (ii) nodes denoted $(e_{i1} , N_i) , \cdots , (e_{ir_i} , N_i)$ if edges $e_{i1} , \cdots , e_{ir_i}$ enter N_i in $\bar{G}$, for $i = 1, 2, \cdots , n + 1$.

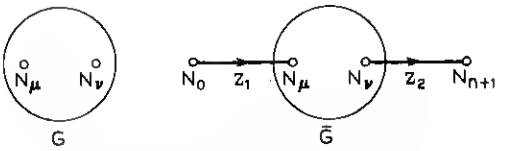
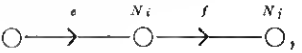
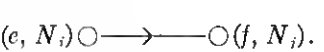


Fig. 9—Construction of augmented graph $\bar{G}$.

The edges of G^T are (i) an edge from (N_o) to $(z_1\ ,\ N_\mu)$, labeled (z_1); and (ii) for every edge-edge transition in $\bar{G}$,



there is a corresponding edge in G^T:



In general, we see that nodes of G^T have labels of the form (edge of $\bar{G}$, node of $\bar{G}$), and edges have labels of the form (edge-edge transition pair of $\bar{G}$).

By construction, apart from the edge (z_1) of G^T, there is a one-to-one correspondence between edge-edge transitions in $\bar{G}$ and edges of G^T.

To find $\gamma_{\mu\nu}$ we apply algorithm B to G^T. Each path through G^T from N_o to N_{n+1} will have the form

$$(z_1),\ (z_1e_{i_1}),\ (e_{i_1}\ ,\ e_{i_2}),\ (e_{i_2}\ ,\ e_{i_3}),\ \cdots\ ,\ (e_{i_r}z_2) \tag{3}$$

and this corresponds uniquely to the path

$$e_{i_1}\ ,\ e_{i_2}\ ,\ \cdots\ ,\ e_{i_r} \tag{4}$$

from N_μ to N_ν in G. The process of obtaining (4) from (3) will be called *contracting*.

We can now state the algorithm.

Algorithm C for Obtaining $\gamma_{\mu\nu}$

1. From G obtain $\bar{G}$ and then the transition graph G^T.

2. Apply algorithm B to find a small set of spanning paths from N_o to N_{n+1} in G^T.

3. Contract each of these paths to give a set of paths in G. This is $\gamma_{\mu\nu}$.

Example: Let G be the graph of Fig. 6. Then $\bar{G}$ and G^T are shown in Figs. 10–11.

Applying algorithm B, or in this case even by inspection, we see that a minimal spanning set for Fig. 11 is

$$\begin{aligned}
&(z_1)(z_1a_1)(a_1b_1)(b_1c_1)(c_1z_2)\\
+\ &(z_1)(z_1a_1)(a_1b_2)(b_2c_1)(c_1z_2)\\
+\ &(z_1)(z_1a_2)(a_2b_1)(b_1c_2)(c_2z_2)\\
+\ &(z_1)(z_1a_2)(a_2b_2)(b_2c_2)(c_2z_2),
\end{aligned}$$

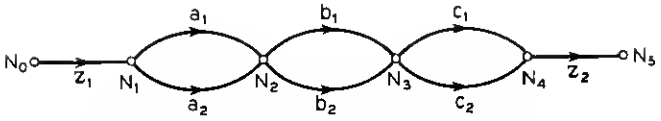


Fig. 10—Augmented graph $\tilde{G}$ corresponding to Fig. 6.

which contracts to give the paths

$$a_1b_1c_1 + a_1b_2c_1 + a_2b_1c_2 + a_2b_2c_2 ,$$

the same solution as found before.

VII. FINDING THE MOST PROBABLE PATHS

A directed graph G is given with a conditional probability measure associated with the edges. More precisely G has nodes $N_1 , \cdots , N_n$, and associated with each edge e, directed say from N_i to N_j , is the conditional probability p_e that e will be traversed next, given that the last node reached was N_i .

We wish to find the most probable paths through the graph, starting at N_μ and ending at N_ν . The probability of a path is the product of the probabilities associated with the edges in the path.

In other words it is desired to find those paths P for which

$$\text{probability } (P) = \prod_{\substack{\text{all edges} \\ e \in P}} p_e$$

is the maximum, or is close to the maximum.

If we label each edge e of G with the "length"

$$q_e = -\log p_e$$

instead of with p_e , an equivalent problem is to find those paths P for which

$$\sum_{\text{all edges } e \in P} q_e$$

is the minimum, or is close to the minimum. In the new graph this corresponds to finding the shortest paths between N_μ and N_ν . This problem has been extensively studied and many good algorithms for its solution are available. We refer the reader to the recent survey by S. E. Dreyfus.[15] References 16 and 17 are earlier surveys covering a wide range of similar problems. The paper by H. Frank[18] is also relevant.

VIII. SUMMARY

Four questions which arise in testing a stored program for possible errors are stated quite generally in terms of listing the paths through

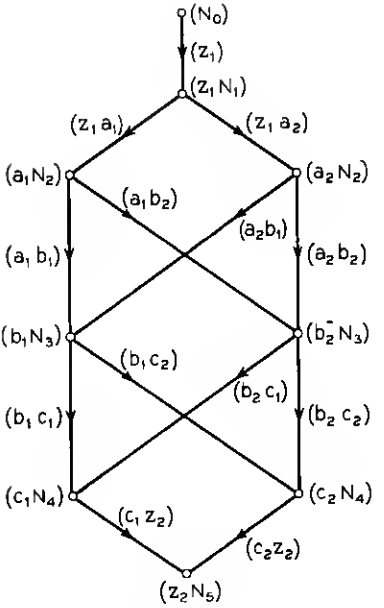


Fig. 11—Transition graph G^T corresponding to Fig. 10.

a directed graph. Question 1 may be answered for small graphs by the McNaughton–Yamada algorithm, and for large graphs by partitioning (Section III). Question 2 involves a difficult combinatorial problem, the minimal covering problem, a partial solution of which is given by the appropriately named greedy algorithm of Section IV. With the aid of the greedy algorithm, algorithm B solves question 2 (Section V). Question 3 is solved by the same method as question 2 (Section VI). Question 4 is shown to be equivalent to the widely studied "shortest-path problem," and references are given to the appropriate literature (Section VII).

IX. ACKNOWLEDGMENT

The author wishes to thank J. M. Scanlon for many interesting discussions and helpful suggestions.

REFERENCES

1. Harary, F., *Graph Theory*, Reading, Mass: Addison-Wesley, 1969.
2. Scanlon, J. M., personal communication.
3. McNaughton, R., and Yamada, H., "Regular Expressions and State Graphs for Automata," IRE Trans. Elec. Computers, *EC-9*, No. 1 (March 1960), pp. 39–47.

4. Hennie, F. C., *Finite-State Models for Logical Machines*, New York: Wiley, 1968.
5. Griswald, R. E., Poage, J. F., and Polonsky, I, P., *The SNOBOL4 Programming Language*, Englewood Cliffs, New Jersey: Prentice-Hall, 1968.
6. Mirsky, L., and Perfect, H., "Systems of Representatives," J. Math. Anal. and Appl., *15*, No. 3 (September 1966), pp. 520–568.
7. Breuer, M. A., "Simplification of the Covering Problem with Application to Boolean Expressions," J. Assoc. Comp. Mach., *17*, No. 1 (January 1970), pp. 166–181.
8. Cobham, A., Fridshal, R., and North, J. H., "An Application of Linear Programming to the Minimization of Boolean Functions," AIEE 2nd Annual Symposium on Switching Theory and Logical Design, 1961, pp. 3–10.
9. Geoffrion, A., "Integer Programming by Implicit Enumeration and Balas' Method," SIAM Review, *9*, No. 2 (April 1967), pp. 178–190.
10. House, R. W., Nelson, L. D., and Rado, T., "Computer Studies of a Certain Class of Linear Integer Problems," in *Recent Advances in Optimization Techniques*, edited by A. Lavi and T. Voge, New York: Wiley, 1966.
11. Lawler, E. L., "Covering Problems: Duality Relations and a New Method of Solution," J. SIAM Appl. Math., *14*, No. 5 (September 1966), pp. 1115–1132.
12. Mayoh, B. H., "On Finding Optimal Covers," Int. J. Comp. Math., *2*, No. 1 (January 1968), pp. 57–73.
13. McCluskey, E. J., Jr., "Minimization of Boolean Functions," B.S.T.J., *35*, No. 6 (November 1956), pp. 1417–1444.
14. Roth, R., "Computer Solutions to Minimum-Cover Problems," Operations Research, *17*, No. 3 (May–June 1969), pp. 455–465.
15. Dreyfus, S. E., "An Appraisal of Some Shortest-Path Algorithms," Operations Research, *17*, No. 3 (May–June 1969), pp. 395–412.
16. Fulkerson, D. R., "Flow Networks and Combinatorial Operations Research," Amer. Math. Monthly, *73*, No. 2 (February 1966), pp. 115–138.
17. Hu, T. C., "Recent Advances in Network Flows," SIAM Review, *10*, No. 3, (July 1968), pp. 354–359.
18. Frank, H., "Shortest Paths in Probabilistic Graphs," Operations Research, *17*, No. 4 (July–August 1969), pp. 583–599.